MULTIMEDIA UNIVERSITY

STUDENT ID NO

MULTIMEDIA UNIVERSITY

FINAL EXAMINATION

TRIMESTER 2, 2017/2018 SESSION

ECE3206 – OBJECT ORIENTED PROGRAMMING WITH C++

(CE, EE, ME, TE)

17 MARCH 2018
2:30 P.M – 4:30 P.M.
(2 Hours)

INSTRUCTIONS TO STUDENT

1. This question paper consists of eight (8) pages only (including this page).
2. **This is an open-book exam,** so the questions are more in-depth and there are fewer questions than normal exams. Students are allowed a maximum of 5 stapled pages of reference sheet into the exam hall.
3. There are **THREE (3) QUESTIONS** in this paper. Answer **ALL QUESTIONS.** Question 1 carry 50 marks and Question 2 and 3 each carries 25 marks.
4. All programming questions must be answered using the C++ language.
5. Write your answers in the Answer Booklet provided.
6. State all assumptions clearly.

Question 1

Figure Q1-1 illustrates a Unified Modeling Language (UML) class diagram for class **`Rainfall`**. This class reads and stores the data readouts of rain (in mm) for each day of the week, calculates the highest, lowest and average points in the data set, and displays them for the user to see.

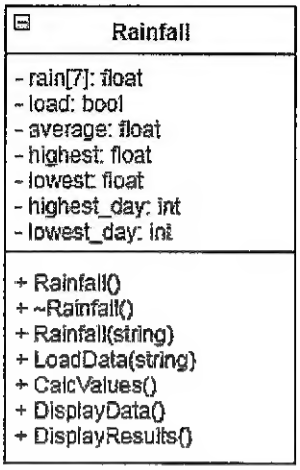

Figure Q1-1: Class `Rainfall` UML diagram.

Figure Q1-2 and Figure Q1-3 illustrate the two files `week1.txt` and `week2.txt` containing the data of the rainfall for two separate weeks.

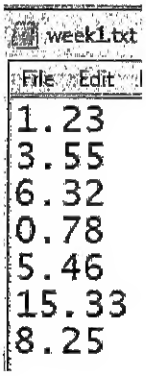

Figure Q1-2: `week1.txt`

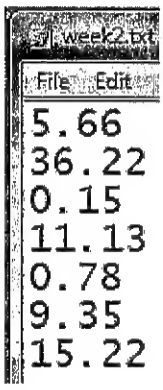

Figure Q1-3: `week2.txt`

Using C++ programming language and with reference to Figures Q1-1:

(a) Implement all the member functions, constructor and destructor of class **`Rainfall`**.

i. Declaration of the private members of the class according to the UML diagram.

[3 marks]

ii. The **`Rainfall()`** default constructor sets the private members to their default values of zero (including all elements in the `rain[]` array), and false for Boolean variable `load`.

[3 marks]

iii. The **`Rainfall(string fileName)`** parameterized constructor uses the input **`fileName`** parameter to load the file. The constructor runs **`LoadData()`** method which is described below and upon success runs the **`CalcValues()`** followed by **`DisplayData()`** and then **`DisplayResults()`** methods. Success in opening and loading the data is

determined by the status of the member Boolean variable `load`, which is set to `TRUE` if the data is successfully loaded or `FALSE` otherwise. If the data file is not successfully opened, the constructor outputs the error "`Unable to open file. Please check for the correct file name and reload.`". Sample output of both successful and unsuccessful instantiations of the **`Rainfall`** class using parameterized constructor are shown in Figure Q1-4 and Figure Q1-5 respectively.

```
week1.txt:Data loaded successfully.
Day 1:1.23MM
Day 2:3.55MM
Day 3:6.32MM
Day 4:0.78MM
Day 5:5.46MM
Day 6:15.33MM
Day 7:8.25MM
Highest rainfall was 15.33mm at Day 6
Lowest rainfall was 0.78mm at Day 4
Average rainfall was 5.84571mm.
Press any key to continue . . .
```

Figure Q1-4: output from Rainfall(String Filename) constructor upon success

```
fail.txt:Data not loaded successfully.
Unable to open file. Please check for the correct the file name and reload.
Press any key to continue . . .
```

Figure Q1-5: output from Rainfall(String Filename) constructor upon failure

[6 marks]

iv. The **`LoadData(string fileName)`** member function takes in a string called `fileName` as input parameter. It then uses the fstream `readFile()` method to open the file.

If the file can be opened, it will read in up to 7 lines of data representing the daily rainfall for the week. The input lines from the file are read as a string for each line before being converted to a float variable using `stof()` and then stored in the array `rain[]`. Upon success of opening the file, it will close the fstream data file, set the Boolean member variable `load` to TRUE and print out the success message as in Figure Q1-6. If the file cannot be opened, it will print the error message as in Figure Q1-7.

[Hint: use `stof()` function to convert `string` variables to floating point values.]

Continued ..

```
week1.txt:Data loaded successfully.
Press any key to continue . . .
```

Figure Q1-6: output from LoadData() method upon success

```
fail.txt:Data not loaded successfully.
Press any key to continue . . .
```

Figure Q1-7: output from LoadData() method upon failure

[10 marks]

v. If `load` is `true`, **`CalcValues()`** member function calculates the highest, lowest and average rainfall stored in member array `rain[]`. This function throws a string `"No data loaded yet."` if it is called while the value of the Boolean member variable `load` is false.

[10 marks]

vi. The **`DisplayData()`** member function prints out all the elements in the `rain[]` array in the format shown in Figure Q1-8 . It throws the error string `"No data loaded yet."` if the `load` member variable is false.

```
Day 1:1.23MM
Day 2:3.55MM
Day 3:6.32MM
Day 4:0.78MM
Day 5:5.46MM
Day 6:15.33MM
Day 7:8.25MM
Press any key to continue . . .
```

Figure Q1-8: sample output when DisplayData() is called.

[Hint: the `rain[]` array starts counting from index 0, whereas **`DisplayData()`** shows the days starting from Day 1.]

[2 marks]

vii. The **`DisplayResults()`** member function display highest, lowest and average value of the rainfall data as shown in Figure Q1-9. It throws the similar error string `"No data loaded yet."` if the `load` member variable is false.

[3 marks]

```
Highest rainfall was 15.33mm at Day 6
Lowest rainfall was 0.78mm at Day 4
Average rainfall was 5.84571mm.
Press any key to continue . . .
```

Figure Q1-9: sample output when DisplayResults() is called.

[Hint: the `rain[]` array starts counting from index 0, whereas **`DisplayResults()`** shows the days starting from Day 1.]

Continued..

(b) Write a **main()** function to test the implemented class **Rainfall** with proper exception handling. The sample output of the entire program is displayed in Figure Q1-10.

i. Declare and initialize an object for class Rainfall called `week1` using the parameterized constructor. Use the data file `"week1.txt"` as the input parameter.

[3 marks]

ii. Declare an object for class **Rainfall** called `week2` using the **default** constructor. Then, test exception handling by calling the **CalcValues()** method in a **try-catch** block without calling **LoadData()** beforehand.

[5 marks]

iii. Continue the `main()` function from part (b)(ii) by calling **LoadData()** method with `week2.txt` as input parameter. Subsequently call **CalcValues()**, **DisplayData()** and **DisplayResults()** methods for the `week2` object within a **try-catch** block.

[5 marks]

```
week1.txt:Data loaded successfully.
Day 1:1.23MM
Day 2:3.55MM
Day 3:6.32MM
Day 4:0.78MM
Day 5:5.46MM
Day 6:15.33MM
Day 7:8.25MM
Highest rainfall was 15.33mm at Day 6
Lowest rainfall was 0.78mm at Day 4
Average rainfall was 5.84571mm.

LoadData() for week 2 not called.
No data loaded yet.

week2.txt:Data loaded successfully.
Day 1:5.66MM
Day 2:36.22MM
Day 3:0.15MM
Day 4:11.13MM
Day 5:0.78MM
Day 6:9.35MM
Day 7:15.22MM
Highest rainfall was 36.22mm at Day 2
Lowest rainfall was 0.15mm at Day 3
Average rainfall was 11.2157mm.
Press any key to continue . . .
```

Figure Q1-10: Complete sample output for part (b) main function.

Note: *Use C++ file processing when reading from the text file. Apply the necessary* **try** *and* **catch** *procedures in the* **main()** *function.*

Continued ...

Question 2

a) Implement a class named **Movies** with the following properties.
Data members: `title (string), director (string), duration (integer) and quality` (integer indicating 0 to 5 stars)
Methods: `setTitle, setDirectorName, setDuration, setQuality, printOutput`
Make the **`printOutput`** method polymorphic in order to use it to print the data for each created objects in the **Movies** hierarchy. Implement all member methods and data members in order to produce the display similar to Figure Q2-1.

Before a movie is released, some scenes are deleted by the director. Derive a class **DirectorNotCut** from **Movies.** This class store the movie that contain the undeleted and deleted scenes. Include new data members to hold the original time (`oriTime`) and the changes (`changes`). Implement the accessor method (`setChanges, setOriTime`) to store the respective newly added information in the member variables, and also the corresponding method (`printOutput`) to display the data.

Derive a class **ForeignMovies** from **Movies**. Add a data member (`language`) to hold the language. Implement the required method (`setLanguage`) to store the language in the data member, and also the corresponding method (`printOutput`) to output the data.
Write a **`main()`** function to use the polymorphic method `printOutput` to print the following screen in Figure Q2-1: The member method and data variable name mentioned in the question must be used.

```
Movies--
Title: Hidden Arrow
Director: Alfred Hitchcock
Duration: 112 mins
Quality: ****

DirectorNotCut--
Title: Good student
Director: Ed Wood
Duration: 70 mins
Quality: **
Original time: 172 mins
Changes: Extra footage not in original included

ForeignMovies--
Title: Bonjour
Director: Francois Truffaut
Duration: 104 mins
Quality: ****
Language: French
```

Figure Q2-1

[20 Marks]

b) Draw the UML class diagram to represent the relationship between the three classes declared in question 2 (a). [5 Marks]

Continued ...

Question 3

a) Use the code given in Figure Q3-1 as your reference. Implement C++ code for all the methods in the Stack template class. The method should operate with any numeric data types (e.g. float, int, double). Use the given `main()` driver function as your reference. The `main()` function stores the number 1, 2, 3, 4, 5 into the stack and then pop them back in reverse order.

```
#include <iostream>
#include <cstdlib>
#define default_value 10
using namespace std;

template< class T > class Stack
{
    public:
    Stack(int = default_value);//default constructor
    ~Stack()//destructor
    {delete [] values;}
    bool push( T );
    T pop();
    bool isEmpty();
    bool isFull();
    private:
    int size; // store maximum size of stack
    T *values;
    int index;

};

int main()
{
    Stack <double> stack1;
    int i, j;
    cout << "\n pushed values into stack1: ";

    for( i = 1  ; i <= 5 ; i++)   {
    if(stack1.push(i))
        cout << endl << i;
    else
        cout << "\n Stack1 is full: ";
    }

    cout << "\n\n pop values from stack1 : \n";
    for( j = 1 ; j <= 5 ; j++)
    cout << stack1.pop() << endl;

    return 0;
}
```

Figure Q3-1

[12 Marks]

Continued ...

b) Implement a template class (`Calculator`) and its member template methods to support the following driver function. The method `displayResult()` shall call the template method `add()`, `subtract()`, `multiply()` and `division()` that return the result of the arithmetic operation on different types of number. The returned result should be displayed.

[13 Marks]

```
int main()
{
 Calculator<int> intCalc(2, 1);
 Calculator<float> floatCalc(2.4, 1.2);
 cout << "Int results:" << endl;
 intCalc.displayResult();
 // display result of 2+1,2-1,2*1,2/1
 cout << endl << "Float results:" << endl;
 floatCalc.displayResult();
 // display result of 2.4+1.2, 2.4-1.2, 2.4*1.2, 2.4/1.2
 return 0;
}
```

End of Paper